No. 2017-1118 & No. 2017-1202

IN THE

UNITED STATES COURT OF APPEALS
FOR THE FEDERAL CIRCUIT

ORACLE AMERICA, INC.,

Plaintiff-Appellant,

v.

GOOGLE INC.,

Defendant-Cross Appellant.

ON APPEAL FROM THE UNITED STATES DISTRICT COURT FOR THE NORTHERN DISTRICT OF CALIFORNIA, CASE NO. 10-CV-3561 HON. WILLIAM H. ALSUP

BRIEF OF SCOTT MCNEALY AND BRIAN SUTPHIN AS *AMICI CURIAE* IN SUPPORT OF REVERSAL

STEVEN T. COTTREAU
CLIFFORD CHANCE US LLP
2001 K STREET, NW
WASHINGTON, DC 20006
(202) 912-5000
Steve.Cottreau@
CliffordChance.com

Counsel of Record

Counsel for Amici Curiae

February 17, 2017

CERTIFICATE OF INTEREST

Pursuant to Federal Rules of Appellate Procedure 26.1 and 47.4, Counsel for *Amici Curiae* certifies the following:

1. The full name of every party or Amici Curiae we represent is: Scott McNealy and Brian Sutphin

2. The name of the real party in interest represented by us is: None.

3. All parent corporations and any publicly held companies that own 10 percent or more of the stock of the party or amicus curiae represented by us are: None.

4. The following law firm and partners or associates have appeared or are expected to appear before this Court on behalf of the Amici Curiae are:

STEVEN T. COTTREAU
CLIFFORD CHANCE US LLP
2001 K STREET, NW
WASHINGTON, DC 20006
(202) 912-5000
Steve.Cottreau@
CliffordChance.com
Counsel of Record

Dated: February 17, 2017

By: /s/ Steven T. Cottreau
STEVEN T. COTTREAU

TABLE OF CONTENTS

TABLE OF AUTHORITIES

Cases

Other Authorities

Rules

Trial & Deposition Testimony

INTEREST OF AMICI CURIAE

Amici Curiae Scott McNealy and Brian Sutphin ("Sun Executives") are former executives of Sun Microsystems, Inc. ("Sun"), who were integrally involved in the development and widespread adoption of the Java platform. Sun, which was founded in 1982, developed the world's most innovative products and services, which have been used to power the world's key computing systems. Through its commitment to shared innovation, community development, and open source leadership, Sun quickly became a leader in the sale of computer workstations. In the 1990s, Sun developed Java, an object-oriented, platform-independent, multithreaded programming environment that revolutionized computer programming and quickly became the foundation for the World Wide Web along with numerous computing and networking devices.

Amicus curiae Scott McNealy co-founded Sun and was the Chairman of its Board of Directors from 1984 to 2010, President from December 1984 to April 1999 and from July 2002 to April 2004, and Chief Executive Officer from December 1984 to April 2006. In these roles, Mr. McNealy worked to make Sun an innovative leader in the information-technology industry.

Amicus curiae Brian Sutphin joined Sun in 1994 and from 2004 through 2010 was Sun's Executive Vice President of Corporate Development and Alliances. Mr. Sutphin's responsibilities included mergers and acquisitions,

creating corporate-level alliances with key global technology companies, and establishing policies and requirements for Sun's inbound technology licensing.

In 2010, Oracle Corp. ("Oracle") acquired Sun, and the Sun Executives moved on to other projects. While the Sun Executives do not have any current involvement with Oracle or Java, they share a vital interest in protecting Java's creative legacy.

The Sun Executives, because of their close involvement with the creation and widespread adoption of Java, feel compelled to submit this brief to explain that Java was the result of exceptional creative effort and Google's copying of Java's APIs was not a fair use. Consequently, the jury verdict must be overturned.[1]

[1] Pursuant to Federal Rule of Appellate Procedure 29(a), both parties have consented to the filing of this brief. No party to this case or its counsel authored this brief in whole or in part, and no person other than amici and their counsel made a monetary contribution to its preparation or submission. *See* Fed. R. App. P. 29(c)(5).

ARGUMENT

The Java platform revolutionized the software industry for computers, smartphones, video games, ATMs, and thousands of other devices. It is the foundation upon which our digital world is built. Google stole that foundation, used it to build Android, and destroyed Oracle's market in the process.

No reasonable jury could have found that Google's copying of Java's APIs for Android satisfied any of the four factors constituting fair use under the Copyright Act. Google did not transform the Java API packages in any meaningful way. Instead, it used them for the same purposes on platforms on which Java was already operating. Android's use is entirely commercial. The copied API packages are the heart and soul of Java, and Google's copying eviscerated the market for Java. Moreover, Android's unauthorized use of the copied APIs was entirely unnecessary, as demonstrated by other industry players.

To truly understand the breadth and depth of this unfair use, one must understand what the Java framework is, why it was created, and the platforms on which it has operated. That exposition informs the reason why Google's fair use defense must fail as a matter of law.

I. OVERVIEW OF JAVA FRAMEWORK

A. Java Was Created As A Platform Neutral System

Prior to the release of the Java platform, the dominant programming languages permitted developers to use a few common rules and vocabularies in writing programs for different types of computer systems and devices. Each device, however, had its own unique requirements. Thus, prior to the advent of the Java platform, each program had to be written in a manner that was specific to a particular device. Although a programmer could write programs for multiple systems or devices in the same language, the program itself would have to be re-written (or "ported") in a manner that was specific to each type of computing device. For example, an application written in the programming language C and designed for a Microsoft Windows PC would not work on a phone, tablet, or on any other non-Windows device.

Porting programs for multiple systems was massively inefficient and often prohibitively expensive. Even if a software developer was willing to incur the expense, the process took valuable time and thus slowed the process of making software available for new platforms and devices. As a result, developers often

chose to write software for systems with the largest number of users. Early efforts to develop a cross-system platform failed.[2]

Innovators at Sun realized they needed to start "really [bearing] down and … help customers solve the problems they were having in migrating away from mainframes."[3] A team of Sun computer engineers, led by James Gosling, began developing a computer programming platform intended to revolutionize how people would program.

The Java platform was the result: a paradigm-shifting platform that permitted developers to "**Write Once, Run Anywhere**." Software developers could write a program once, using Java, and the program could run on a variety of different computing systems and devices.

Java's "Write Once, Run Anywhere" promise was an inspirational creative breakthrough in software development. Unlike programs written on predecessor programming platforms, a program written and developed once with the Java development platform would work on a large selection of systems and devices.[4]

---

[2] James Daly, *Apple, Symantec Rethink Role Bedrock Will Play*, Computerworld, Dec. 20, 1993, at 69 (describing failure of early effort at programming language designed to run in Apple and Microsoft environments)

[3] Lee Gomes, *Made in the shade; 'Java' stirs up renewed interest in Sun Micro*, The Dallas Morning News, Dec. 18, 1995.

[4] David Bank, *The Java Saga*, Wired Magazine, Dec. 1995.

Central to the Java platform's success was its inclusion of a collection of creatively pre-written programs bundled into "packages" (also known as APIs or Application Programming Interfaces) that allowed programmers to code quickly and efficiently. At issue in this case are the choice of what packages to create, the creativity of their naming and organization, and the value and importance of those packages to Java.

Due to its elegant and robust design, the Java platform proved an enormous success. The Java platform has evolved and thrived for more than 20 years while competing against myriad other notable development platforms. At present, Java runs on 15 billion devices worldwide, with more than 5 million people studying the programming language and 10 million using it to develop next generation programs.[5] Internet-defining companies, such as Twitter and Netflix, rely on Java for the infrastructure to power their businesses.[6] Java's success not only advanced "Write One, Run Anywhere," but it was at the forefront of the World Wide Web revolution of the 1990s, "bring[ing] the Internet to a new level of experience for all users."[7] Moreover, as early was 2000, Research in Motion—the company that

---

[5] Java Powers Our Digital World, https://go.java/index.html?intcmp=gojava-banner-java-com (last visited Feb. 16, 2017).

[6] *Id.*

[7] Elizabeth Corcoran, *Java Jumps Into the 'Net'; Proponents Say New Software Language Could Herald Computing Revolution*, The Washington Post, Dec. 10, 1995.

created the Blackberry smartphone—was using the Java platform to create a programmable, web-enabled pager.[8] The Java platform opened up the world of mobile phones and devices to the possibilities of the internet.[9]

B. How Java Works

The Java platform relies upon five primary elements: (i) a Java programming language with which developers can write applications; (ii) a core set of programs (interchangeably called the Java Packages, Java APIs, or Java Class Library) which developers can use to speed the creation of new applications; (iii) a Java compiler, which translates the code written by the developer into Java byte-code; (iv) a Java Virtual Machine ("JVM"), which translates Java byte-code into instructions comprehensible to the underlying computing platform or device; and (v) a Java Development Kit ("JDK"), a collection of programming tools released by Oracle. If the Java Class Library and a JVM are present on a system or device, the system is said to carry a "Java Runtime Environment" and that system can run applications written in Java. Of these five elements of Java, only Google's copying of the Java Packages (i.e., APIs) is at issue in this case.

1. ***Java Language.*** The Java language constitutes the "bare bones" of the Java framework. The language provides the syntax, grammar, and vocabulary

---

[8] Mark Beaulieu, *Wireless Internet Applications and Architecture: Building Professional Wireless Applications Worldwide* 289 (2002).

[9] *See id.*

of the language to permit a software developer to write programs in Java code that will run in a Java runtime environment.

2. ***Java Packages.*** Java provides an extensive set of packages organized into a library. These packages are at issue in this case. The Java Packages are an extensive set of ready-to-use programs that serve as helpful "building blocks" for Java developers. Sun developed these pre-built packages to allow programmers to accomplish programming tasks ranging from the simple (such as basic math functions) to the complex (such as providing computer security and network access functions). In other words, Java developers need not "reinvent the wheel" for many desired programming tasks that were commonly used by a wide variety of computer programs. Rather, the creators of the Java platform included these functions in the Java Packages.

Although there are many intricacies to how the Java Packages interact with one another and within themselves, a package is generally subdivided into classes or interfaces, which are further subdivided into methods. Each method contains the discrete programming functions used by developers. For example, the java.net package provides 40 classes and interfaces to implement networking applications (*i.e.*, connecting to the internet and other computer networks). It contains 440 methods that range from determining if the computer or device is connected to a

locally networked computer, to retrieving the IP address of another computer connected to the internet.

3. ***Java Compiler.*** The Java compiler reads and interprets the source code written by a developer, including its declaring code to the Java Packages, and generates a more compact Java byte-code. This byte-code is what is distributed to end-users to run on different computing platforms.

4. ***Java Virtual Machines ("JVM").*** JVMs are the final link to Java's "run anywhere" platform. Installed on a user's device, the JVM is a piece of software that translates the Java byte-code to enable it to run on each particular computer or device. In short, to run Java programs on a particular computer platform, the JVM must be customized for that platform. Once a JVM exists for a platform, all devices using that type of platform can run programs written in Java.

5. ***Java Development Kit ("JDK").*** The JDK is the culmination of all the elements of the Java platform. It provides all the necessary programs and tools to develop and test Java applications, including the Java Packages, the Java Compiler, and the JVM. By providing all the pieces, the widespread distribution of the JDK was fundamental to promoting Java's "Write Once, Run Anywhere" vision.

C. Early Cross-Platform Uses Of Java

As early as 1991, Sun was looking to the future of mobile devices and the intersection of digital devices and everyday life. Sun established the "Green Team," a group of 13 people initiated by Patrick Naughton, Mike Sheridan, and James Gosling. The team was tasked with anticipating and planning for the "next wave" in computing.[10] It quickly developed the Star7, a PDA mobile device with an animated touch screen that ran an early version of what would become the Java platform.[11] It could control multiple entertainment devices and appliances, such as televisions, VCRs, and stereos, running on a "processor-independent" language, the precursor of Java's JVM environment.[12]

From these early beginnings, Java was born as a method by which developers could exploit the possibilities of the internet across a wide variety of connected devices. The Java platform transformed what had been a medium for sending text and images to a vibrant, multimedia environment with nearly infinite possibilities. Within one year of its release of the Java platform in 1995, Sun had 38 paying licensees and 6,000 developers attending its trade show.[13] Within three

---

[10] Jon Byous, *Java Technology: An Early History* 1 (1998), http://srjcstaff.santarosa.edu/~dpearson/mirrored_pages/java.sun.com/Java_Technology_-_An_early_history.pdf.

[11] Java Timeline, *1992*, http://oracle.com.edgesuite.net/timeline/java/.

[12] Byous at 1.

[13] *Id.* at 8.

years, Java had 150 licensees, including such major players as IBM and Netscape, with hundreds of thousands of developers worldwide.[14] In 1997, the Java platform was already expanding beyond the internet and more traditional desktop PCs to items such as smartcards.[15]

In 1998, *amicus* McNealy, in his role as Sun CEO, explained that "[t]he Java platform is dwarfing any other API or programming environment out there. Java has become the language, the platform, and the architecture for computing on the network."[16]

Notably, mobile devices and cross-platform utility were part of the platform's raison d'être from the very beginning. At 1998's JavaOne Developer Conference, attendees received a JavaRing, a wearable device with an embedded microprocessor. The JavaRing used the Java platform to bring attendees coffee customized to their personal preferences after being scanned by readers located throughout the conference. The demonstration showed Java's utility in a far reaching array of applications and device form factors, many of which are now routine parts of smartphones, such as exchanging contact information, using banking services, and starting cars.[17]

---

[14] *See id.* at 9.

[15] Java Timeline, *1997*.

[16] Byous at 9.

[17] Java Timeline, *1998*.

This focus on mobile devices continued in 1999 with the introduction of the Java 2 Platform, Micro Edition ("J2ME"), a derivative of Java's Standard Edition ("J2SE") designed specifically for mobile devices.[18] Years before Google considered creating Android, Sun had developed an entire infrastructure for the use of the Java platform in mobile computing devices, including the Java Phone API.[19] By the year 2000, Java was ubiquitous in the growing personal communications market, appearing in two-way pagers, mobile phones, and palm computers/PDAs.[20]

A demonstration at the JavaOne conference in 2002 showcased a mobile phone-controlled robot in a sumo wrestling match with a desktop PC-controlled robot.[21] The demonstration's message was clear: Java, designed as a cross-platform system, enabled mobile devices to do anything a computer could do. As mobile devices, including smartphones and tablets, grew more sophisticated and had greater processing power, developers realized they could use the Java platform to bring programs that were previously imprisoned in large desktop machines to sleek, portable handheld devices.[22]

Five years later, Google released Android.

---

[18] *Id.*, *1999*.

[19] *Id.*, *2000*.

[20] *See id.*, *2000*.

[21] *Id.*, *2002*.

[22] Trial Tr. ("Tr.") 1670:9-13 (Testimony of Alan Brenner, Sr. Vice-President, Sun Client Systems Group).

Before the District Court, Google argued that Android was a fair use of the duplicated Java API packages, because Java was not a part of the smartphone market. That was not true.

II. ANDROID IS NOT TRANSFORMATIVE

Google's copying of the Java API packages was simply not transformative. As noted below, both Java and Android targeted smartphones, Android did not use Java's APIs in a "new context," and, in any event, today's powerful modern mobile devices are similar to the PCs on which Java also operated.

A. Java And Android Were Used For The Same Purpose At The Same Time In The Same Context

A work is transformative if it does not "merely supersede the objects of the original creation … [but] instead adds something new, with a further purpose or different character, altering the first with new expression, meaning, or message."[23] Google directly copied the Java API declarations at issue verbatim and without alteration, and used them for the same purpose in Android. In short, Google did nothing more than "supersede the objects of the original creation" without adding any new elements.

Google appropriated Java's copyrighted code specifically because it would allow it to tap into the large existing base of developers already familiar with the

[23] *Campbell v. Acuff-Rose Music, Inc.*, 510 U.S. 569, 579 (1994).

Java APIs.[24] The only way Google's theft could have succeeded was by making sure that the Java APIs in Android were *exactly* the same, accomplishing the same function using the same language. The court below stated "of course, the copied [APIs] serve the same function in both works," Java and Android, "for by definition, declaring code in the Java programming language serves [a] specific definitional purpose."[25] Thus, the District Court recognized this point, but not its import: had Google truly transformed the Java APIs, they would not have had the same function using the same language in the same context and Google would not have been able to steal the legions of developers already using the Java platform.

Nor did Google's theft move Java's APIs into a new mobile context. At trial, the jury heard undisputed evidence that Java was at the heart of the burgeoning smartphone industry in the early and mid 2000s. One of the first, if not the first, smartphones—the Danger Sidekick—ran on Java. Its creator, Andy Rubin, admitted his company "created our own implementation of the Java 2 SE APIs" and obtained a Java license from Sun.[26]

---

[24] *See* Tr. 629:24-30:2 (Testimony of Andy Rubin, Android Creator and Google Employee) ("Rubin Testimony") (noting the use of software already created was "a huge accelerant to our effort" to create Android).

[25] *Oracle America, Inc. v. Google Inc.*, No. C 10-03561, 2016 WL 3181206, at *8 (N.D. Cal. June 8, 2016).

[26] Tr. 887:23- 889:3 (Rubin Testimony).

After licensing Java for use on smartphones, Rubin went on to found Android, which was then acquired by Google.[27] Rubin testified that Android considered Java a direct competitor that was "**targeting the same industry with similar products**."[28] In ruling that a reasonable jury could determine that Google transformed Java, the District Court failed to give adequate weight to this critical admission. Rubin, the man who created Android, admitted that Sun/Oracle was already in the smartphone industry; that the product Android was offering was attempting to accomplish the same goals; and that Android used Java's APIs verbatim.

B. Java Was Designed To Be Device And Platform Independent

Google argued below that Android constituted fair use because it expanded the Java framework into a new area (smartphones), thus adding to and transforming the work to create something new. As noted above, Java quickly expanded to mobile devices before Android was created. In fact, Java was always intended to work across different kinds of hardware. As one Java founder put it, "it was patently obvious" as early as 1993 that Java "fit perfectly with the way applications were written, delivered, and used on the Internet."[29]

---

[27] Tr. 624:11-17 (Rubin Testimony).

[28] Tr. 844:21-22 (emphasis supplied).

[29] Byous at 4.

The first instance of Java was not even created for a PC—it was created for a handheld device, the Star7.[30] Google's Rubin testified that the "rest of the industry … was using Java in some of the phones."[31] As a result, Rubin felt it was important to have a Java license when he launched his own smartphone, the Danger Sidekick.[32] Java was used in a variety of pre-Android devices, including the Blackberry, HTC, Nokia, Danger, and SavaJe.[33] *Amicus* McNealy personally negotiated a Java ME license with Motorola, which was, at the time, perhaps the biggest player in the mobile phone industry.[34]

Thus, contrary to mobile phones being "a fresh context," the Java platform already existed in the "operating environment of mobile smartphone devices" before the advent of Android.[35] In fact, due to its elegance, portability, and functionality, Java dominated the mobile market, including both feature phones[36] (Java ME) and emerging smartphones (Java ME and SE).[37]

---

[30] *Id.* at 2.

[31] Tr. 913:7-9 (Rubin Testimony).

[32] *Id.*

[33] Tr. at 1622:13-21; 1623:10-1624:1 (Testimony of Neal Civjan, frm. VP Worldwide OEM Software Sales, Sun Microsystems).

[34] *See* Tr. 1356:3-9 (Testimony of Safra Catz, CEO, Oracle) (noting Motorola among major Java licensees).

[35] *Oracle America, Inc.*, 2016 WL 3181206, at *8.

[36] "Feature phones" are devices that bridge the gap between pure cell phones, allowing only voice calls and text messaging, and smartphones, containing

The Java platform's use in mobile was by design. The Java platform was meant to unlock the potential of the internet and mobile computing, giving developers the opportunity to learn one language that could be used across many devices. Indeed, without the Java platform, the internet itself, let alone the universe of devices that connect to the internet, would be a very different place. The internet had an early major challenge: many different devices sought to access a given website. Websites are, after all, nothing more than computer programs existing on a remote computer known as a server. Java enabled these computer programs (i.e., websites) to run on any device running a Java Virtual Machine, regardless of the type of device or operating system.[38] The Java platform met this goal by creating a rich ecosystem, closely guarded and nurtured by Sun and, later, Oracle, designed to ensure cross-platform compatibility—Java's core "Write Once, Run Anywhere" value.

---

advanced functionality more akin to a desktop PC. *See Feature phone*, PC Magazine Encyclopedia, http://www.pcmag.com/encyclopedia/term/62894/.

[37] Java ME was in 79 percent of wireless handsets, constituting over 600 different types of phones using networks provided by over 180 carriers in 2005, years before Android launched. Trial Ex. 134 at 3. In 2005, 612 million wireless phone units, representing 79 percent of all units shipped, were pre-installed with Java. *Id.*

[38] Edward Yourdon, *Java and the new Internet programming paradigm*, JavaWorld (Aug. 1, 1996), http://www.javaworld.com/article/2077231/java-and-the-new-internet-programming-paradigm.html.

Android did nothing other than duplicate Sun/Oracle's copyrighted expression. That is not transformative; it is mere duplication.

C. The Java API Packages Were Used In The Same Way On Smartphones

In addition to serving the same purpose in Android, Sun and Oracle always expected that Java platform would expand its presence in mobile devices as the processing power of these devices advanced. Google's copying of Java did nothing new. The Java platform was designed for all devices possessing a threshold level of processing power. Early mobile phones and other resource constrained devices ran Java ME—a slimmed-down derivative of Java SE. Illustrating Java's adaptation to advancing device capabilities, some early smartphones, such as the first Sidekick (which had a black and white screen, a handful of applications, and a primitive web browser) could run Java SE. [39]

Modern smartphones have processing power, graphics, applications, and browsers more advanced than the PCs that first ran Java. Smartphones today—many of which run Android—have a fully integrated email and messaging system, the speed and complexity of which makes mid-2000s PC and phone versions look quaint. The smartphone is no different in processing power from the PCs that were

[39] Klint Finley, *Tech Time Warp of the Week: Before the iPhone, Anyone Who Was Anyone Rocked a Sidekick*, Wired, May 2015, https://www.wired.com/2015/05/tech-time-warp-week-iphone-anyone-anyone-rocked-sidekick/.

available when Android launched. The only difference is that smartphones today fulfill Java's long-standing goal of bringing extensive computing power and digital awareness to the masses in handheld devices.

* * *

In short, Android copied Java's APIs without engaging in any transformative addition. The copied APIs perform the same functions in the same way for the same reason. Android targeted mobile devices—a space where Java's APIs were already present and, indeed, were dominant. Moreover, the way in which Android used the copied APIs was identical to the way they were used in PCs. No reasonable jury could have found that Google transformed Java in any way.

III. THE OTHER FAIR USE FACTORS ALSO FAVOR ORACLE

The other three factors in the fair use analysis—the nature of the work, amount and substantiality of the taking, and the effect on the potential market for and value of the work —also weigh against a fair use finding.

A. Android's Nature Is Commercial

It is black-letter copyright law that "commercial as opposed to nonprofit [use] is a separate factor that weighs against a finding of fair use."[40] Google uses Android commercially. Google could have purchased a license from Oracle, and

[40] *Campbell*, 510 U.S. at 585 (quoting *Harper and Row Publishers, Inc. v. Nation Enterprises*, 471 U.S. 539, 562 (1985)).

the jury heard from Android's creator, Rubin, that Google engaged in extensive discussions to do just that before releasing Android.[41] Ultimately, Google decided to copy Java, not license it. Google's failure to respect Java's intellectual property crystallizes why Google's use was not fair: "The crux of the profit/nonprofit distinction is not whether the sole motive for the use is monetary gain but whether **the user stands to profit from exploitation of the copyrighted material without paying the customary price**."[42]

Nothing would be wrong with Google choosing a different business model from Oracle. Oracle made revenue from Java through licensing fees paid by those seeking to use the Java platform in their devices and programs.[43] Google chose a different method: it gave away the Android source code for free, but Google made its money through advertising and other search functions embedded in the phones and devices using Android.[44] In doing so, Google locked out Java compatible programs. It used the Java APIs, verbatim—saving Google the expense of creating its own APIs and educating its own base of Android developers—but Google changed the surrounding programs just enough to make certain that programs that

---

41 *See* Tr. 886:3-888:5 (Rubin Testimony).

42 *Harper and Row Publishers*, 471 U.S. at 562 (emphasis supplied).

43 Tr. at 1771:17-19 (Testimony of Adam Jaffe, Oracle Economics Expert) ("Jaffe Testimony").

44 *Id.* at 1771:20-23.

could run on Android could not run in the Java Virtual Machine. In fact, Android was designed intentionally so that it would be ***incompatible*** with the Java Virtual Machine, effectively cutting Oracle out of the potential Android market.[45]

A clearer example of failing to pay the fair price for the use of copyrighted material in order to exploit that material for commercial gain is difficult to imagine.

B. Google Copied The Java Platform's Soul

As the Supreme Court noted in *Harper & Row Publishers, Inc. v. Nation Enterprises*, even if "[i]n absolute terms, the words actually quoted were an insubstantial portion of" the copyrighted work, if those words are "essentially the heart of the [work]" and "play a key role in the infringing work," this factor weighs against a fair use finding.[46]

Here, the Java APIs Google copied for Android are the very heart of Java. Google's own Principal Engineer (who previously worked as Senior Sun Engineer

---

[45] *See* Tr. 1332:1-2 (Testimony of Prof. John C. Mitchell, Oracle Programming Expert) ("Mitchell Testimony") ("So you don't really have compatibility. You can't ship code from one platform to another."); Tr. at 1440:25-1441:2 (Testimony of Edward Screven, Chief Corporate Architect, Oracle) ("Screven Testimony") (noting Android's programming "locks programmers into Android … their applications can't run in other environments, other than Android."); Ed Bott, *The Real History of Java and Android, as told by Google*, Sept. 8, 2011, http://www.zdnet.com/article/the-real-history-of-java-and-android-as-told-by-google/ (noting Android was designed to be incompatible with Google).

[46] 471 U.S. at 564-66.

on the Java project), Joshua Bloch, stressed the importance of the creative process of APIs. He explained, "APIs can be among a company's greatest assets" and integral to the process of package design is naming:

> The names that you come up with for classes and for methods sort of—they're talking to you …. They should come out nicely. They should work nicely together. You know, good names can drive development.[47]

As a result, "[n]ames matter a lot": if a programmer creates elegant names, "then your code will kind of read like prose."[48] He has stressed the critical element of literary creativity in naming and organizing packages: "API design is an art, not a science. Strive for beauty, and trust your gut."[49]

As a leading Java educator has explained, "[w]hat has impressed me most as I have come to understand Java is that somewhere in the mix of Sun's design objectives, it seems that there was a goal of reducing *complexity for the programmer*."[50] Indeed, countless programmers have noted the creative design of the Java platform and its packages. As one programmer has put it,

> When I first began to program in Java, I loved the Java language a lot. I used to program in Pascal, Delphi,

---

[47] Joshua Bloch, *How to Design a Good API & Why it Matters*, Javapolis Conference, Nov. 21, 2006, http://www.infoq.com/presentations/effective-api-design.

[48] *Id.*

[49] Joshua Bloch, *Bumper-Sticker API Design*, InfoQ, Sept. 22, 2008, http://www.infoq.com/articles/API-Design-Joshua-Bloch.

[50] Bruce Eckel, *Thinking in Java* 1 (4th ed. 2006).

> Visual Basic and C but Java was very different and elegant. In addition to its language structure and features, its API set was very special. With its beautiful and aesthetic design, programming in Java is a pleasure. I don't have this feeling when I program in other languages. To feel pleasure or pain is also valid when we use API sets. There are many API sets we use in any development cycle coming from different frameworks or libraries. API beauty depends on designer knowledge and design capability (say artistic skill).[51]

As a result of the creative selection, naming and organization of these packages, Java has become one of the most enduring programming platforms ever conceived due to its creative design and elegant organization.

As Oracle's opening brief explains, Google's infringement also is a quantitatively large portion of the Java API packages. But even if it were not, in the same way the magazine writer in *Harper and Row Publishers* copied a work's "dramatic focal points,"[52] the copying of Java's APIs appropriated the creative and elegant essence of Java—its organization, structure, and malleability. This factor weighs heavily against a finding of fair use.

---

[51] Ibrahim Levent, *Beautiful API Design*, DZone, Nov. 26, 2008, http://java.dzone.com/news/beautiful-api.

[52] At issue in *Harper & Row* was a magazine reporter's verbatim copying of segments of a forthcoming autobiography of President Gerald Ford in which Ford discussed his rationale behind pardoning President Richard Nixon. 471 U.S. at 542-43.

C. Google's Infringement Devastated The Market For Java

The jury heard indisputable evidence that Android simply devastated the market for Java.[53] Prior to Android's launch, Java was "in over 85 percent of the [mobile phone] market."[54] Virtually every mobile phone on the market ran some version of Java.[55] After Android's launch, Android "was being adopted in terms of new design wins for phones across the board and displacing Java on those phones and having a massive impact very quickly."[56] Google's business model of giving away Android's source code to "lock in" software developers to the Android system if they wanted to design apps for a wide array of phones effectively destroyed Java's licensing model. Sun's Neal Civjan noted Android "hijacked" Java: Google "took our technology and they gave it away for free and they took our customers and it was devastating."[57] For instance, Motorola eventually dropped its license for Java in favor of Google's free Android.[58]

Ultimately, Java found itself unable to compete with a free version of its own product. Former licensees were able to obtain for free, from Google, the same

---

[53] *See, e.g.*, Tr. at 1639:23-25 (Civjan Testimony) (Q: "Can you tell us, sir, in one word the impact of Android on Java?" A: "It was devastating.").

[54] Tr. at 1624:21-24 (Civjan Testimony).

[55] *See* Trial Ex. 134 at 3.

[56] Tr. at 1632:6-15 (Civjan Testimony).

[57] Tr. 1641:5-17.

[58] Tr. 1356:3-9 (Catz Testimony).

API packages performing the same functions on the same platform for which they previously had to pay. As a result, Oracle eventually had to exit the smartphone market entirely, falling from a dominant position to a non-existent position in less than a decade.[59]

This was not simply the result of market forces at work. This was Google's calculated strategy to subvert the Java platform's market position by taking Oracle's technology, giving it away for free, and generating a revenue stream in a different manner such that no rational customer would remain with Oracle. Had Google created its own technology, its actions might be cutthroat but effective business. By stealing Oracle's longstanding copyrighted material, however, Google unfairly destroyed the market for Java.

IV. GOOGLE'S COPYING OF ORACLE'S APIS WAS NOT NECESSARY

The district court seemed to credit—and, by extension, held that a reasonable jury could credit—Google's argument that Android had no choice but to use Java's APIs, because use of Java's APIs and declaratory code represented the only way code could be called upon to operate. In other words, Google

[59] Tr. at 1361:23-1362:2 (Catz Testimony); Depo Tr. at 135:23-136:4 (Deposition Testimony of Larry Ellison, Chairman of the Board, Oracle) ("Ellison Testimony").

purportedly had to use Java's APIs because no other way existed for Android to do what Java did.

For example, Android had to have a command to activate the code for setting a time zone to allow its code to operate on smartphones. So did Java. But Google did not **have to copy** Java's unique and creative method for each of the functions in the Java API. Taking the Time Zone example, the Java API allowed a Java programmer to access the "DateFormat" class of the java.text package and declare the "setTimeZone" method.[60] By just looking at their names, a developer will intuitively know that the DateFormat class can be used to format a date, and then use the setTimeZone method to set the actual time zone for that developer's application. In Android, Google's copying permitted developers to use the exact same language to accomplish the exact same object in the exact same platform as Java creatively created.

Contrary to the district court's reasoning, however, creators of competing computer programming environments can accomplish this same function in an unlimited number of different creative ways. Apple's iOS platform devotes an

[60] DateFormat (Java 2 Platform SE 5.0), API Specification, http://docs.oracle.com/javase/1.5.0/docs/api/java/text/DateFormat.html.

entire class to set the time zone in an application—the "NSTimeZone" class.[61] Unlike Java's placement of that package in the java.text package, Apple put it in its "Foundation" framework. A framework is Apple's terminology for a structure conceptually similar to Java's "package." Apple's NSTimeZone class contains numerous methods to manipulate time zones, including, retrieving time zones with abbreviations ("timeZoneWithAbbreviation"), retrieving time zones with names ("timeZoneWithName"), and setting the default time zone ("setDefaultTimeZone").[62] It was Apple's creative decision to organize the time zone programs in this manner, select time zone programs that it believed was desirable to programmers, and name the time zone programs as they chose.

Likewise, Microsoft provides similar functionality, but with an entirely different structure, naming scheme, and selection. In its Windows Phone development platform, Microsoft stores its time zone programs in the "TimeZoneInfo" class in its "System" namespace (Microsoft's version of a "package" or "framework").[63] Within that organizational structure, Microsoft has programs to, among other things, convert time from different time zones

---

[61] NSTimeZone Class Reference, Mac Developer Library, https://developer.apple.com/library/mac/#documentation/Cocoa/Reference/Foundation/Classes/NSTimeZone_Class/Reference/Reference.html.

[62] *Id*.

[63] TimeZoneInfo Class (System), Windows Phone Dev Center, http://msdn.microsoft.com/en-us/library/windowsphone/develop/system.timezoneinfo(v=vs. 105).aspx.

("ConvertTime") or determine whether a particular date and time in a particular time zone is ambiguous ("IsAmbiguousTime").[64]

Ultimately, it was the creativity and elegance of Java that caused its adoption and once-dominant market share. As an analogy, there are many paths one can follow to climb a mountain. Some are more difficult than others, some provide different challenges, and some provide better vistas. Yet, at the end, each path leads to the same place, the top of the mountain. The district court essentially held that there was only one conceivable path up the mountain—the creative Java path created by Sun/Oracle. Microsoft and Apple (and many other language developers) had no trouble forging their own paths; only Google chose to unlawfully take Java's.

CONCLUSION

When Sun developed the Java platform, it unified disparate computer systems and devices globally under its "Write Once, Run Anywhere" programming platform. Sun, and later Oracle, invested a great deal of time and resources into the Java platform at great monetary risk. By using the Java API packages without a license, Google stole Oracle's work for its own commercial gain. In pursuit of Android dominance, Google's pure copying was non-transformative, was entirely

[64] *Id*.

commercial, stole the heart of the Java system, and devastated the market for Java. No reasonable jury could find that Google's use of the copied APIs was fair.

Dated: February 17, 2017

Respectfully submitted,

By: /s/ Steven T. Cottreau
STEVEN T. COTTREAU
CLIFFORD CHANCE US LLP
2001 K STREET, NW
WASHINGTON, DC 20006
(202) 912-5000
Steve.Cottreau@
CliffordChance.com
Counsel of Record

Counsel for Amici Curiae Scott McNealy and Brian Sutphin

CERTIFICATE OF COMPLIANCE WITH FED. R. APP. P. 29 AND 32

1. This *Amici Curiae* Brief complies with the type-volume limitations of Federal Rules of Appellate Procedure 29(d) and 32(a)(7)(B) because it contains 6,828 words, excluding the parts of the Brief exempted by Fed. R. App. P. 32(a)(7)(B)(iii).

2. This *Amici Curiae* Brief complies with the typeface requirements of Federal Rules of Appellate Procedure 32(a)(5) and the type style requirements of Rule 32(a)(6) because this has been prepared in a proportionally spaced typeface using Microsoft Word 2013 14-point Times New Roman font.

Dated: February 17, 2017

/s/ Steven T. Cottreau
Steven T. Cottreau (Counsel of Record)
Counsel for Amici Curiae

CERTIFICATE OF SERVICE

I hereby certify that on February 17, 2017, I electronically filed a copy of this *Amici Curiae* Brief of Scott McNealy and Brian Sutphin in Support of Plaintiff-Appellant with the Clerk of the Court for the United States Court of Appeals for the Federal Circuit using the Appellate CM/ECF system, which will automatically send email notification of such filing to all counsel of record:

Dated: February 17, 2017

/s/ Steven T. Cottreau
Steven T. Cottreau (Counsel of Record)
Counsel for Amici Curiae